La redondance dans les CSPs

Assef Chmeiss, Vincent Krawczyk, Lakhdar Saïs

La redondance dans les CSPs

Assef Chmeiss **Vincent Krawczyk** **Lakhdar Saïs**

Université Lille-Nord de France, Artois, F-62307 Lens
CRIL, F-62307 Lens
CNRS UMR 8188, F-62307 Lens
{chmeiss, krawczyk, sais}@cril.univ-artois.fr

Résumé

Dans ce papier, une nouvelle technique pour calculer des sous-ensembles irredondant de réseaux de contraintes est proposé. La vérification d'une contrainte donnée est connue pour être Co-NP complet. Dans notre approche, différentes consistances locales polynomiales sont utilisées pour réduire la complexité. Le réseau de contraintes obtenu est irredondant *modulo* une consistance locale donnée. Notre principal objectif est de mesurer les relations entre la redondance des contraintes et l'efficacité des solveurs CSP. Plus précisément, les contraintes redondantes sont éliminées de l'instance d'origine produisant un problème équivalent par rapport à la satisfiabilité. Notre approche admet des particularités intéressantes. Premièrement, l'élimination de contraintes redondantes peut aider le solveur CSP à diriger la recherche vers la partie la plus contrainte du réseau. Deuxièmement, une telle élimination peut conduire à améliorer le comportement des heuristiques de choix de variables et à réduire le nombre de vérification de contraintes. De façon intéressante, notre approche peut être utilisée pour mesurer le degré de redondance des contraintes des instances CSPs. Les résultats expérimentaux sur les instances prises de la dernière compétition CSP montre clairement l'intérêt de l'approche que nous proposons.

1 Introduction

Les problèmes de satisfaction de contraintes (CSPs) impliquent l'assignation de valeurs à des variables qui sont soumises à un ensemble de contraintes. L'approche CSP est maintenant largement utilisée pour modéliser et résoudre de nombreux problèmes industriels tels que la planification et la configuration. Les phases de modélisation et de résolution sont connues pour être fortement liées. Ainsi, l'efficacité des solveurs dépend de la manière dont le problème est modélisé. Jusqu'à récemment, ces deux phases étaient considérées séparément. De nombreuses améliorations ont été proposées pour la résolution et d'autres approches ont suggéré de simplifier l'étape cruciale de modélisation [4, 9, 7]. Cela est clairement important pour l'augmentation de l'audience de la technologie des contraintes. Plus généralement, le pricipal challenge est de trouver un réel compromis entre la simplicité de la modélisation du problème et l'efficacité des solveurs CSP. Comme il existe de nombreuses manières de modéliser le même problème, cema signifie que l'utilisateur peut introduire des redondances dans le processus de modélisation. Des redondances peuvent aussi résulter de l'encodage incorrect ou la fusion de différentes parties de nombreuses sources. Le réseau de contraintes obtenu peut contenir des parties qui peuvent être supprimées sans perdre l'information qu'elles contiennent. Cependant plusieurs formes de redondances peuvent être caractérisées.

Dans ce papier, nous nous intéressons aux redondances de contraintes. Un CSP est redondant si et seulement si certaines de ses contraintes peuvent être supprimées tout en préservant son ensemble de modèles. Comme établi par Paolo Liberatore [8] dans le contexte de formules clausales propositionnelles, ce problème est clairement important pour plusieurs raisons. Premièrement, la suppression de contraintes redondantes peut simplifier le réseau de contraintes en réduisant sa taille. Une grande partie des redondances peuvent complexifier le véritable ensemble de contraintes (la partie irredondante du réseau de contraintes) qui code le problème. Dans d'autres cas, la redondance peut indiquer que certaines parties du réseau de contraintes sont plus importantes que les autres. Par conséquent, dépendant du domaine d'application, la redondance peut être un concept soit positif soit négatif.

Notre principal objectif est de mesurer les relations entre la redondance de contraintes et l'efficacité des solveurs

CSP. Notre approche peut être vue comme une technique possible de vérification du degré de redondance d'un réseau de contraintes donné. Sur les instances CSPs disponibles, notre approche peut donner une façon d'approximer la partie irredondante du problème. Cependant, la vérification de la redondance des contraintes, c'est à dire décider si une contrainte donné peut être déduite du reste du problème, est connu pour être Co-NP complet[8]. Différentes consistances locales polynomiales sont utilisées pour réduire la complexité du calcul des contraintes redondantes. Le réseau de contraintes obtenu est irredondant *modulo* une consistance locale donnée.

Le reste du papier est organisé comme suit. Après quelques définitions et notations préliminaires une nouvelle consistance locale utilisé pour la vérification des contraintes redondantes est donnée dans la section 3 et 4. La section 5 présente la redondance dans les instances CSP et le concept de redondance *modulo* une consistance locale est développé. Dans la section 6, nos résultats expérimentaux sont présentés et analysés. Finalement, des remarques conclusives sont données dans la dernière section.

2 Définitions et notations préliminaires

Un CSP (problème de satisfaction de contraintes) est défini comme un tuple $\mathcal{P} =< \mathcal{X}, \mathcal{C} >$. $\mathcal{X}$ est un ensemble fini de n variables $\{x_1, x_2, \ldots, x_n\}$. Chaque variable $x_i \in \mathcal{X}$ est défini sur un ensemble fini de d_i valeurs, noté $dom(x_i) = \{v_{i_1}, v_{i_2} \ldots v_{i_{d_i}}\}$. $\mathcal{C}$ est un ensemble fini de m contraintes $\{c_1, c_2, \ldots, c_m\}$. Chaque contrainte $c_i \in \mathcal{C}$ d'arité k est défini comme un couple $(scope(c_i), R_{c_i})$ où $scope(c_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq \mathcal{X}$ est l'ensemble des variables impliquées dans c_i et $R_{c_i} \subseteq dom(x_{i_1}) \times \ldots \times dom(x_{i_k})$ l'ensemble des tuples autorisés, i.e. $t \in R_{c_i}$ si et seulement si le tuple t satisfait la contrainte c_i. Nous notons $t[x]$ la valeur de la variable x dans le tuple t. Un CSP $\mathcal{P}$ est dit binaire si et seulement si $\forall c_i \in \mathcal{C}, |scope(c_i)| \leq 2$. Pour plus de simplicité, une contrainte c avec $scope(c) = \{x_i, x_j\}$ sera noté c_{ij}.

Une interprétation est une assignation d'une valeur pour chaque variable $x \in \mathcal{X}$. Un modèle (solution) est une interprétation qui satisfait toutes les contraintes. La résolution d'un CSP $\mathcal{P}$ implique soit de trouver un modèle ($\mathcal{P}$ est satisfaisable) soit de prouver qu'il n'existe pas de modèle ($\mathcal{P}$ est insatisfaisable). Le solveur CSP le plus populaire est un algorithme de recherche en profondeur d'abord avec retour arrière, où à chaque étape de la recherche, une assignation de variable est faite suivie par un processus de filtrage appelé propagation de contraintes. Usuellement, les algorithmes de propagation de contraintes, qui sont basés sur des propriétés des réseaux de contraintes tel que la consistance d'arc, suppriment des valeurs qui ne peuvent pas apparaitre dans une solution. L'algorithme qui maintient la consistance d'arc durant la recherche est appelé MAC[10].

Dans le reste du papier, nous limitons notre présentation aux CSPs binaires. Cependant, notre approche peut être facilement étendue aux CSPs n-aire même si les contraintes sont représentées en intention à condition de pouvoir calculer facilement la négation de ces contraintes. Nous rappelons ci-dessous des définitions de consistances locales qui seront utiles dans le reste du papier.

Definition 1 (Consistance d'arc (AC)) *Soit $\mathcal{P}$ un CSP. Une variable $x_i \in \mathcal{X}$ est arc-consistante ssi, $\forall v \in dom(x_i)$, $\forall x_j \in \mathcal{X}$ tel que $c_{ij} \in \mathcal{C}$, il existe $b \in dom(x_j)$ tel que $(a, b) \in R_{c_{ij}}$. Un CSP est arc-consistant ssi $\forall x \in \mathcal{X}, dom(x) \neq \emptyset$ et $dom(x)$ est arc-consistant.*

La consistance d'arc peut être généralisée pour les CSP n-aire et est appelé consistance d'arc généralisée[2].

Definition 2 (Consistance d'arc généralisée (GAC)) *Soit $\mathcal{P}$ un CSP. Une variable $x \in \mathcal{X}$ est arc consistante généralisée (GAC) ssi, $\forall v \in dom(x)$, $\forall c \in \mathcal{C}$ tel que $x \in scope(c)$, il existe un tuple valide $\tau \in R_c$ tel que $v = \tau[x]$. Un CSP est GAC ssi $\forall x \in \mathcal{X}, dom(x) \neq \emptyset$ et $dom(x)$ est GAC.*

Une autre consistance locale plus puissante que AC est la consistance d'arc singleton (SAC)[5]. Elle peut identifier plus de valeurs inconsistantes que AC. SAC garantit que l'application de AC après avoir effectué une assignation d'une variable à une valeur ne conduit pas à l'insatisfaisabilité. Des notations sont nécessaires pour définir formellement cette consistance locale forte. Nous définissons $\phi(\mathcal{P})$ comme le CSP $\mathcal{P}$ obtenu après l'application d'une consistance locale ϕ. For $\phi = AC$, cela signifie que toutes les valeurs arc inconsistantes sont supprimées de $\mathcal{P}$. S'il y a une variable avec un domaine vide dans $\phi(\mathcal{P})$, nous notons $\phi(\mathcal{P}) = \perp$. Le sous-réseau obtenu après l'assignation d'une variable x à une valeur v est notée $\mathcal{P}|_{x=v}$. La définition suivante décrit SAC.

Definition 3 (Consistance d'arc singleton (SAC)) *Soit $\mathcal{P} =< \mathcal{X}, \mathcal{C} >$ un CSP. Un couple (x, v) tel que $x \in \mathcal{X}$ et $v \in dom(x)$ est SAC ssi $AC(\mathcal{P}|_{x=v}) \neq \perp$. La variable x est SAC ssi $\forall v \in dom(x)$, (x, v) est SAC. $\mathcal{P}$ est SAC ssi $\forall x \in \mathcal{X}$, x est SAC.*

3 Au delà de SAC

Pour une valeur donnée v dans le domaine de la variable x, SAC vérifie si le problème pour lequel le domaine de x est restreint à l'unique valeur v est AC. Si l'assignation de v à la variable x conduit à un domaine vide lorsque l'on applique AC alors la valeur v est supprimée de $dom(x)$. Dans cette section, nous proposons une nouvelle technique de filtrage, appelée consistance d'arc de tuple (TAC). Cette

consistance locale est introduite pour être exploitée dans notre approche (section 5). L'idée principale est qu'au lieu de fixé une valeuir comme pour SAC, nous fixons un tuple d'une contrainte c, c'est à dire, nous assignons les variables impliquées dans c et nous appliquons AC sur le sous-réseau obtenu. Dans cette section, nous présentons TAC pour les CSPs binaires. Nous montrons dans la section suivante comment étendre cette technique aux CSPs n-aires. La définition suivante décrit formellement ce filtrage.

Definition 4 (consistance d'arc de tuples (TAC)) *Soit $\mathcal{P}$ un CSP. Une contrainte $c_{ij} \in \mathcal{C}$ est tuple arc consistant (TAC) ssi $\forall (a,b) \in R_{c_{ij}}, AC(P|_{x_i=a,x_j=b}) \neq \perp$. $\mathcal{P}$ est TAC ssi $\forall c \in \mathcal{C}$, c est TAC.*

L'algorithme 1, appelé *TAC*, applique TAC sur un CSP $\mathcal{P}$. Pour chaque tuple (a,b) de chaque contrainte c_{ij}, l'algorithme vérifie si ce tuple vérifie la propriété. Cela peut être fait en appliquant un algorithme de consistance d'arc sur le problème après avoir restreint le domaine de x_i et x_j à leur unique valeur a et b, respectivement. Si le sous-réseau, dans lequel x_i et x_j ont tous les deux une valeur, est inconsistant alors le tuple (a,b) est supprimé de sa relation (ligne 6) et nous devons propager cette suppression. L'algorithme TAC retourne *faux* quand une relation devient vide (ligne 9) ce qui signifie que le problème est inconsistant, sinon il retourne *vrai*.

Algorithm 1: TAC : booléen

```
   Input: P =< X, C >
1  repeat
2      changed ← false;
3      for each c_ij ∈ C do
4          for each (a,b) ∈ R_{c_ij} do
5              if AC(P|_{x_i=a,x_j=b}) = ⊥ then
6                  R_{c_ij} ← R_{c_ij} \ {(a,b)};
7                  P ← AC(P);
8                  if P = ⊥ then
9                      return false ; // P is inconsistent
10                 changed ← true;
11 until not changed ;
12 return true;
```

La complexité de *TAC* est liée à la complexité de l'algorithme de consistance d'arc utilisé. Si nous considérons AC2001 [3], sa complexité en temps est en $O(md^2)$ et sa complexité spatiale est en $O(md)$, nous pouvons donner une borne supérieure de la complexité de *TAC*.

Theorem 1 *La complexité en temps dans le pire des cas de l'algorithme TAC est $\mathcal{O}(m^3d^6)$ avec une complexité spatiale de $\mathcal{O}(md)$.*

preuve La boucle *For* (ligne 4) est exécutée au plus md^2 fois. A chaque itération, nous appliquons l'algorithme de consistance d'arc de complexité $\mathcal{O}(md^2)$. Donc, le coût des deux boucles *For* est en $\mathcal{O}(m^2d^4)$. Puisque nous pouvons supprimer au plus md^2 tuples, alors la complexité en temps dans le pire des cas de TAC est en $O(m^3d^6)$. La complexité spatiale de *TAC* est bornée par la complexité de l'algorithme AC qui est en $O(md)$.

Proposition 1 *L'algorithme TAC est correct.*

preuve Il est évident que *TAC* ne supprime pas de tuples qui doivent être maintenus. En effet, un tuple (a,b) est supprimé parce qu'il ne vérifie pas la propriété *TAC*. Cela signifie qu'un domaine vide est apparu pour une variable. Alors le tuple supprimé qui peut être vu comme une instanciation de 2 variables ne peut pas faire partie d'une solution.

Clairement, la complexité en temps de TAC étant élevée, dans notre cadre, TAC est principalement utilisé pour vérifier la redondance d'une contrainte donnée. De plus, dans notre implémentation, pour un tuple donné (a,b) (ligne 4), nous vérifions d'abord si la valeur a est SAC, si ce n'est pas le cas, la valeur est supprimée ; sinon la seconde valeur est vérifiée. Cette stratégie ne change pas la complexité dans le pire des cas mais est pls efficace en pratique.

4 gTAC : TAC pour les CSPs n-aires

Dans la section 3, nous avons présenté la technique de filtrage TAC pour les CSPs binaires. Ce filtrage peut être facilement généralisé aux problèmes n-aires. La consistance d'arc de tuple généralisée (*gTAC*) peut être défini comme suit :

Definition 5 (TAC généralisé (gTAC)) *Soit $\mathcal{P}$ un CSP. Une contrainte $c \in \mathcal{C}$ avec $scope(c) = \{x_1, \ldots, x_k\}$ est gTAC ssi $\forall (v_1, \ldots, v_k) \in R_c, AC(P|_{x_1=v_1,\ldots,x_k=v_k}) \neq \perp$. $\mathcal{P}$ est gTAC ssi $\forall c \in \mathcal{C}$, c est gTAC.*

Pour les CSPs n-aires, ce que nous devons changer par rapport au cas binaire est l'algorithme d'AC utilisé. Par exemple, nous pouvons utilisé GAC [2] qui a une complexité en $O(mk^3d^{k+1})$. Il est facile de voir que la complexité en temps dans le pire des cas de l'algorithme *gTAC* est en $O(m^3k^3d^{3k+1}$ avec une complexité spatiale en $O(mk)$.

5 Redondance de contraintes

La redondance, dans un CSP, apparait lorsque des informations sont présentes plusieurs fois, c'est à dire, un sous-ensemble de contraintes peut être déduit du reste. Pour déterminer si une contrainte c est redondante ou non, nous avons besoin de résoudre le CSP avec la négation de la

contrainte c. Ce problème est connu pour être Co-NP complet [8]. Cependant il est possible de détecter en temps polynomial certaines contraintes redondantes en utilisant un "entailment" *modulo* une consistance locale donnée.

Dans cette section, nous définissons formellement la notion de redondance de contrainte en CSP et nous voyons comment nous pouvons utiliser des filtrages locaux pour détecter certaines contraintes redondantes. Dans les problèmes de satisfaction, la redondance *modulo* la propagation unitaire a été montrée très utile en pratique sur les instances industrielles[6].

Dans un CSP, une contrainte c est redondant si elle peut être déduite des autres contraintes. Un CSP $\mathcal{P}$ est redondant s'il contient un sous-ensemble de contraintes redondantes sinon il est appelé irredondant. On peut formalisé ce concept comme suit. Pour un CSP $\mathcal{P} =< \mathcal{X}, \mathcal{C} >$ et une contrainte $c \in \mathcal{C}$, nous définissons $\mathcal{P} \setminus \{c\}$ comme le CSP $\mathcal{P}' =< \mathcal{X}, \mathcal{C} \backslash c >$. Nous définissons la négation d'une contrainte c, notée $\neg c$, comme la contrainte c' telle que $scope(c') = scope(c)$ et $R_{c'} = \{t | t \in \prod_{\forall x \in scope(c)} dom(x), t \notin R_c\}$.

Definition 6 *Soit $\mathcal{P}$ un CSP et $c \in \mathcal{C}$. c est redondante ssi $\mathcal{P} \setminus \{c\} \cup \neg c$ est insatisfaisable.*

Definition 7 *Soit $\mathcal{P}$ un CSP. $\mathcal{P}$ est redondant ssi $\exists c' \in \mathcal{C}$ telle que c' est redondante. Sinon $\mathcal{P}$ est dit irredondant.*

Cette définition est illustré par l'exemple 1. Dans cet exemple, nous avons un CSP $\mathcal{P}$ avec trois variables, trois valeurs par domaine et des contraintes d'égalité entre les variables. Clairement, nous avons une contrainte redondante, puisque nous pouvons facilement déduire que x est égal à z à partir des deux autres égalités. De plus, dans cet exemple, n'importe quelle contrainte peut être déduite des autres. Donc, nous pouvons supprimer une de ces trois contraintes (par exemple c_1) et par conséquent nous obtenons un CSP irredondant. Nous pouvons aussi noté que, un réseau de contrainte peut admettre beaucoup de sous-réseaux irredondants.

Example 1 *Soit $\mathcal{P} =< \mathcal{X}, \mathcal{C} >$ un CSP tel que $\mathcal{X} = \{x, y, z\}$ avec $dom(x) = dom(y) = dom(z) = \{1, 2, 3\}$ et $\mathcal{C} = \{c_1, c_2, c_3\}$ avec $c_1 : x = y$, $c_2 : y = z$ et $c_3 : x = z$.*

Comme mentionné plus haut, pour vérifier si une contrainte c est redondante, nous avons besoin de résoudre le problème $\mathcal{P} \setminus \{c\} \cup \neg c$ et prouver qu'il est insatisfaisable.

Pour contourner ce problème, nous considérons un algorithme incomplet mais polynomial pour détecter les contraintes redondantes. Au lieu de résoudre le problème, nous appliquons un filtrage local ϕ tel que AC ou TAC. N'importe quel autre consistance locale peut être utilisée.

Vérifier si une contrainte est redondante peut être fait en utilisant une procédure de réfutation. Une contrainte $c \in \mathcal{C}$ est redondante ssi le réseau de contrainte dans lequel c est substitué par sa négation est insatisfaisable (voir la définition ci-dessus). C'est clairement intraitable. C'est pourquoi, nous définissons une forme de réfutation plus faible induisant une forme plus faible de redondance.

Definition 8 *Soit $\mathcal{P}$ un CSP et ϕ une consistance locale. Une contrainte $c \in \mathcal{C}$ est ϕ-redondante ssi $\phi(\mathcal{P} \backslash \{c\} \cup \{\neg c\}) = \bot$.*

De la même manière, un CSP $\mathcal{P}$ est appelé ϕ-redondant (respectivement ϕ-irredondant) ssi il contient (resp. ne contient pas) de contraintes ϕ-redondantes. Prenons de nouveau l'exemple 1. Nous pouvons voir que la redondance de la contrainte c_1 (par exemple) n'est pas AC-redondante mais est TAC-redondante.

Algorithm 2: Calcul d'un réseau de contrainte ϕ-irredondant

Input: $\mathcal{P} =< \mathcal{X}, \mathcal{C} >$
Output: A ϕ-irredundant CSP $\mathcal{P}$

```
1 for each c ∈ C do
2     P' ← P \ {c} ∪ ¬c;
3     if φ(P') = ⊥ then
4         C ← C\c;
```

Dans l'algorithme 2, ϕ peut être remplacé par n'importe quel filtrage de consistance locale tel que AC ou TAC. La complexité de l'algorithme 2 est polynomial et dépend du filtrage utilisé. Si nous utilisons un filtrage de consistance d'arc dont la complexité est $O(md^2)$ alors la complexité en temps de l'algorithme 2 est bornée par $\mathcal{O}(m^2d^2)$.

Notons que l'utilisation d'ordres de contraintes différents dans l'algorithme 2, peut conduire à différents sous-réseaux ϕ-irredundant.

6 Résultats expérimentaux

Dans cette section, nous montrons l'intérêt pratique de notre approche. Nous avons conduit deux types d'expérimentations. Premièrement, nous montrons le pouvoir de réduction en terme de pourcentage de contraintes ϕ-redondates supprimées par notre approche avec ϕ instancié à AC et TAC présenté dans la section 3. La seconde expérimentation a pour but de mesurer le role des contraintes redondantes sur l'efficacité des solveurs CSP.

Ici, nous nous sommes focalisés sur les instances binaires. Notre solveur CSP utilise l'algorithme MAC classique (avec AC8 comme algorithme de consistance d'arc) avec comme heuristique de choix de variables *dom/deg* ou *dom/WDeg*.

Le tableau 1 présente les résultats sur un ensemble représentatif d'instances. Dans ce tableau, nous fournissons

	AC, Red_{AC}		AC, Red_{TAC}		TAC, Red_{AC}		TAC, Red_{TAC}	
nom de l'instance (#var, #ctr)	temps	%	temps	%	temps	%	temps	%
bqwh-15-106 (106, 644)	0,03	572 (11%)	0,16	570 (11%)	0,13	559 (13%)	0,25	555 (14%)
domino-1000-800 (1000, 1000)	123,01	0 (100%)	123,85	0 (100%)	123,06	0 (100%)	123,75	0 (100%)
driverlogw-02c-sat	5,5	1910 (53%)	10,36	1756 (57%)	6,78	1428 (65%)	9,58	1367 (66%)
ehi-85-297-0 (297, 4094)	0,26	4094 (0%)	12,29	776 (81%)	10,72	-	11,25	-
frb30-15-1 (30, 208)	0,01	208 (0%)	5,87	208 (0%)	0,11	208 (0%)	5,88	208 (0%)
knights-15-9 (9, 36)	0,04	36 (0%)	46,11	9 (75%)	0,11	36 (0%)	44,64	9 (75%)
langford-3-9 (27, 369)	0,09	325 (12%)	9.75	317 (14%)	31.17	310 (16%)	31,35	37 (90%)
qcp-20-187-4 (400, 7600)	0,11	1444 (81%)	0,61	1444 (81%)	30,09	1444 (81%)	30,45	1444 (81%)
rlfap-graph1 (200, 1134)	0,15	1134 (0%)	101,83	885 (22%)	964	1134 (0%)	1042,26	522 (54%)
rlfapScens4 (680, 3967)	0,34	1309 (67%)	36,13	873 (78%)	112,95	992 (75%)	114,71	635 (84%)
rlfapScensMod2-f24 (200, 1235)	0,08	1235 (0%)	53,72	716 (42%)	244,26	1235 (0%)	271,38	445 (64%)
rlfapscen11-f10 (680, 4103)	0,476	4103 (0%)	197,742	2954 (28%)	69,827	-	72,548	-

TAB. 1 – Résultats sur les benchmarks de la seconde compétition internationale CSPs.

le pourcentage de contraintes phi-redondantes supprimées. Dans la première colonne l'instance du problème est décrit (nom de l'instance, nombre de variables (#var), nombre de contraintes (#ctr)). Dans les quatres double colonnes suivantes, les résultats obtenus en appliquant une consistance ϕ comme prétraitement et une consistance ϕ pour vérifier la redondance sont donnés. Par exemple, dans la seconde double colonne (AC, Red_{TAC}) signifie que nous appliquons AC en prétraitement et TAC pour vérifier la redondance des contraintes. Pour chaque cas, nous donnons le temps d'exécution (en secondes), le nombre de contraintes restantes et le pourcentage de contraintes ϕ-redondantes supprimées. Les instances résolues (prouvées instaisfaisable), dans la phase de prétraitement, sont indiquées avec un tiret "-" au lieu d'un pourcentage.

Si nous regardons l'instance *domino-1000-800*, nous pouvons voir que toutes les contraintes sont supprimées. En effet, toutes les contraintes deviennent redondantes puisque, après le prétraitement, il n'y a plus qu'une valeur par domaine et l'instance est prouvée satisfaisable. Au contraire, pour certaines instances comme *frb30-15-1*, la technique ne détecte aucune contrainte redondante (c'est le cas pour les instances aléatoires). Pour les instances *bqwh-15-106* et *driverlogw-02c-sat*, nous remarquons qu'un filtrage plus fort comme *TAC* détecte plus de contraintes redondantes que *AC* Cette remarque est confirmée pour d'autres classes comme *ehi-85* et *rlfap-graph1* où la détection des contraintes redondantes par TAC est plus significative qu'avec AC. Nous pouvons voir que, pour les classes *ehi-85* et *rlfap-scen11*, le filtrage *TAC* prouve l'inconsistance durant la phase de préprocessing.

Dans les figures 1 et 2, les résultats correspondant à la résolution des instances binaires sont donnés. Notre objectif ici est de montrer l'effet de la suppression des contraintes redondantes sur l'efficacité des solveurs CSP. Ces figures montrent des résultats comparatifs en terme de temps de recherche entre MAC sur l'instance originale et MAC sur le sous-réseau ϕ-irredondant. Un point en dessous de la diagonale signifie que l'approche représentée sur l'axe des y est meilleur. L'axe des x représente les résultats de la recherche sur l'instance originale alors que l'axe des y donne les résultats après la suppression des contraintes redondantes.

Dans la figure 1, nous montrons l'impact de la suppression des contraintes redondantes en utilisant l'approche *AC-redondante* sur MAC et les heuristiques *dom/deg* (figure de gauche) et *dom/wdeg* (figure de droite). Cette figure fournit le temps d'exécution totale (suppression des contraintes ϕ-redondantes et phase de recherche). Nous pouvons remarquer que, l'élimination des contraintes redondantes améliore l'efficacité des algorithmes de recherche. Nous pouvons aussi noter que les améliorations avec l'heuristique *dom/deg* sont plus significatives qu'avec *dom/wdeg*. Ce n'est pas surprenant puisque *dom/wdeg* dirige la recherche vers la partie la plus contrainte du réseau. Par conséquent l'impact de la suppression des contraintes redondantes est moins important avec *dom/wdeg*. Cependant notre approche obtient des améliorations intéressantes même avec cette heuristique.

Dans la figure 2 nous considérons uniquement l'heuristique *dom/deg* avec la détection des contraintes *TAC-redondantes*. Cette figure présente le temps de recherche (figure de gauche) et le temps total (figure de droite). Nous pouvons remarquer que la suppression des contraintes redondantes peut améliorer significativement l'étape de recherche. Cependant, même si nous considérons le temps total, notre technique est meilleur que l'algorithme de recherche (figure de droite). Donc nous pouvons conclure que notre approche améliore l'efficacité de la recherche.

7 Conclusions

Dans ce papier, une nouvelle approche pour calculer les sous-ensembles irredondants des réseaux de contraintes est proposé. En utilisant des techniques de consistances locales pour la vérification de la redondance ont permis d'obtenir sur de nombreuses instances de classes CSP des réductions significatives dans la taille des réseaux. Le sous-réseau obtenu est irredondant *modulo* une consistance lo-

cale. Notre approche montre clairement que l'élimination de contraintes redondantes est très utile pour les solveurs CSP. Les résultats sont particulièrement impressionnant sur MAC avec l'heuristic dom/deg. De plus, notre nouveau filtrage TAC est puissant pour détecter pour détecter les contraintes redondantes. Certaines classes d'instances sont résolues sans recherche lorsque l'on utilise TAC en prétraitement. En utilisant des techniques de consistance locale généralisées, notre approche peut être exploitée sur n'importe quel type de contraintes dont la négation peut être calculé.

Ces résultats ouvrent des perspectives intéressantes, parmi celles-ci, nous allons étendre la vérification de la redondance non seulement pour éliminer les contraintes redondantes mais aussi pour déduire des informations utiles qui peuvent améliorer les solveurs CSPs.

Références

[1] C. Bessiere and R. Debruyne. Theoretical analysis of singleton arc consistency and its extensions. *Artificial Intelligence*, 172(1) :29–41, 2008.

[2] C. Bessiere and J. Régin. Arc consistency for general constraint networks : preliminary results. In *Proceedings of IJCAI'97*, pages 398–404, 1997.

[3] C. Bessiere and J. Régin. Refining the basic constraint propagation algorithm. In *Proceedings of IJCAI'01*, pages 309–315, 2001.

[4] Christian Bessière, Remi Coletta, Barry O'Sullivan, and Mathias Paulin. Query-driven constraint acquisition. In *Ijcai'2007*, pages 50–55, 2007.

[5] R. Debruyne and C. Bessiere. Some practical filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI'97*, pages 412–417, 1997.

[6] Olivier FOURDRINOY, Eric GREGOIRE, Bertrand MAZURE, and Lakhdar SAIS. Reducing hard sat instances to polynomial ones. In *2007 IEEE international conference on Information Reuse and Integration polynomial ones(IEEE-IRI'07)*, pages 18–23, Las Vegas, aout 2007.

[7] Alan M. Frisch, Christopher Jefferson, Bernadette Martínez Hernández, and Ian Miguel. The rules of constraint modelling. In *Ijcai'2005*, pages 109–116, 2005.

[8] Paolo Liberatore. Redundancy in logic i : Cnf propositional formulae. *Artif. Intell.*, 163(2) :203–232, 2005.

[9] Sarah O'Connell, Barry O'Sullivan, and Eugene C. Freuder. Timid acquisition of constraint satisfaction problems. In *Sac'2005*, pages 404–408, 2005.

[10] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of CP'94*, pages 10–20, 1994.

[11] M.R.C. van Dongen. Beyond singleton arc consistency. In *Proceedings of ECAI'06*, pages 163–167, 2006.

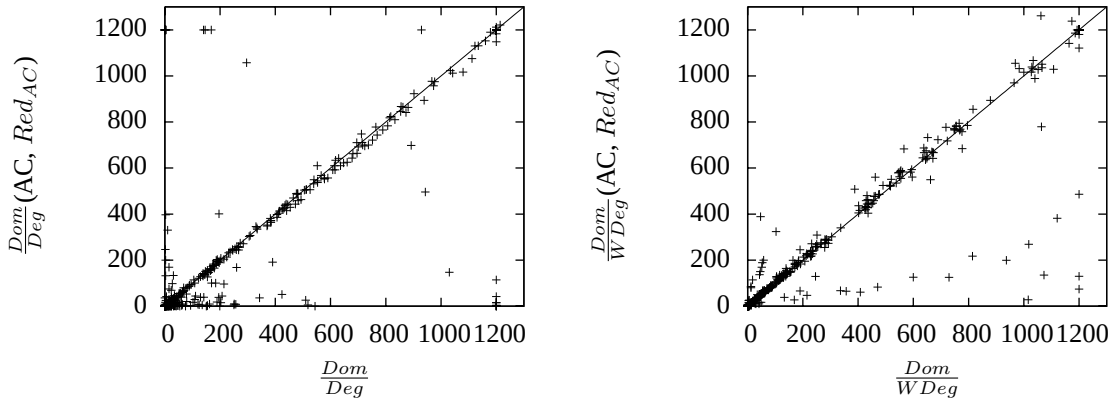

FIG. 1 – Red_{AC} : Redundancy+Search : (dom/deg) Vs (dom/Wdeg)

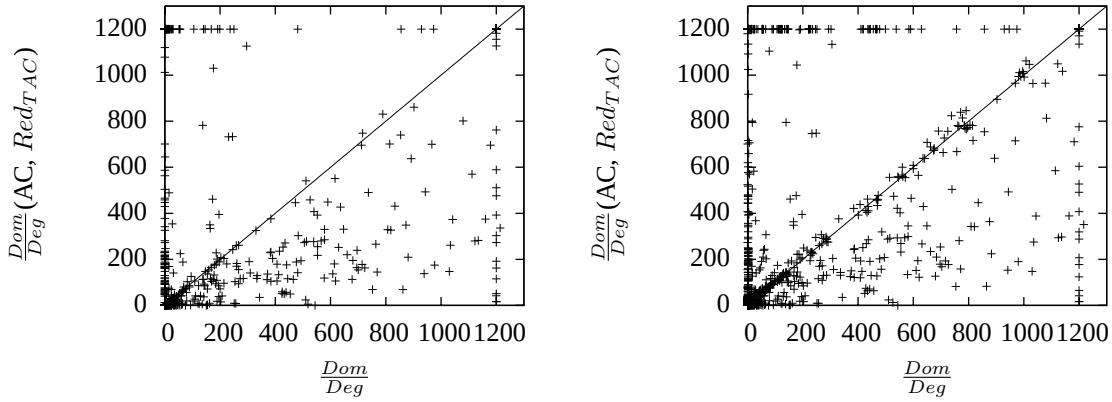

FIG. 2 – Red_{TAC} : Search Vs Redundancy+Search (dom/deg)